TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Automaatika instituut

ISS40LT

MICROSOFT KINECT'IL PÕHINEV KASUTAJALIIDES TORNKRAANALE

Bakalaureusetöö

Üliõpilane: Aleksandr Nikandrov
Üliõpilaskood: 104114 IASB

Juhendajad: Eduard Petlenkov
Dotsent

Aleksei Tepljakov
Insener

Tallinn
2014

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Kuupäev:

Autor: Allkiri:

Declaration

Herewith I declare that this thesis is based on my own work. All ideas, major views and data from different sources by other authors are used only with a reference to the source. The thesis has not been submitted for any degree or examination in any other university.

Date:

Author: Signature:

Microsoft Kinect'il põhinev kasutajaliides tornkraanale

Annotatsioon

Käesoleva töö eesmärgiks on Mikrosoft Kinect-i rakendamine tornkraana kasutajaliidese realiseerimiseks.

Töö põhitulemuseks on tarkvaraliselt realiseeritud rakendus, mis töötab Microsoft Windows platvormil. Kinect sensor on ühendatud rakenduspakettiga MATLAB/Simulink järjestikliidese kaudu. Välja töötatud süsteem annab võimaluse juhtida tornkraana tuvastades operaatori liigutusi.

Kasutajaliides on lihtne ja arusaadav tavakasutajale, kiiresti paigaldatav ja seadistatav. Tööle on lisatud kasutusjuhend (9 lehekülge).

Töö on kirjutatud inglise keeles ja sisaldab 24 lehekülge, 6 peatükki, 8 graafikut.

Microsoft Kinect based HMI for a tower crane

Abstract

The main goal of this work was an attempt to find and implement a real world application of Microsoft Kinect for Windows in real-time control environment.

The result is an application running on a Microsoft Windows based personal computer (PC) communicating with MATLAB/Simulink via serial port. Which allows to control a miniature tower crane with the help of Kinect interaction.

This human machine interface must be simple to use and understand, fast to set up, and easily used with other machinery and/or without the use of MATLAB/Simulink.

The thesis is in English and contains 24 pages of text, 6 chapters and 8 figures.

Table of contents

List of illustrations

1. Introduction

As the motion sensing devices are becoming more sophisticated and popular with each year, one device successfully penetrated the mainstream market – Microsoft Kinect. Manly used as an alternative to controllers in video games, it can be used as powerful tool to augment reality and help controlling dynamic systems that are closer to human movement, rather than analog or digital inputs via analog sticks or buttons.

As the department of computer control has many real-time systems, ranging from 3D crane to inverse pendulum cart, the most obvious choice was a miniature model of an industrial tower crane. The crane has a rotating arm very similar to one humans have. Because of this, moving the trolley, lifting the cargo and rotating the cranes arm can be considered as most intuitive form of gesture control.

Based on the above observations, the main goal of this work was to create and application that will collect raw data from Kinect sensor, operate with it to create control signals that could interpreted and sent to MATLAB. All done on a single Microsoft Windows based PC.

As of current, the application along with Kinect should be used in the laboratory to demonstrate and test the controlled system. Though, if critical problems could not be found, the interface can be operated in any environment, as long as Kinect sensor is secured, in stable position, and not in danger.

The fact that the whole human machine interface (HMI) should be used everywhere by anyone means that interface must be user friendly and the whole setup can be installed with ease. Furthermore, because a tower crane can be very dangerous, the application must have a fast and easy termination sequence with an ability to set default position of the crane.

2. System description

2.1 INTECO Tower Crane

The three-dimensional model of industrial crane is a highly nonlinear MIMO (Multiple Input Multiple Output) system equipped with a dedicated system of 2D angle measuring unit and sensors. The system is fully integrated with MATLAB/Simulink and operates in real-time [1].

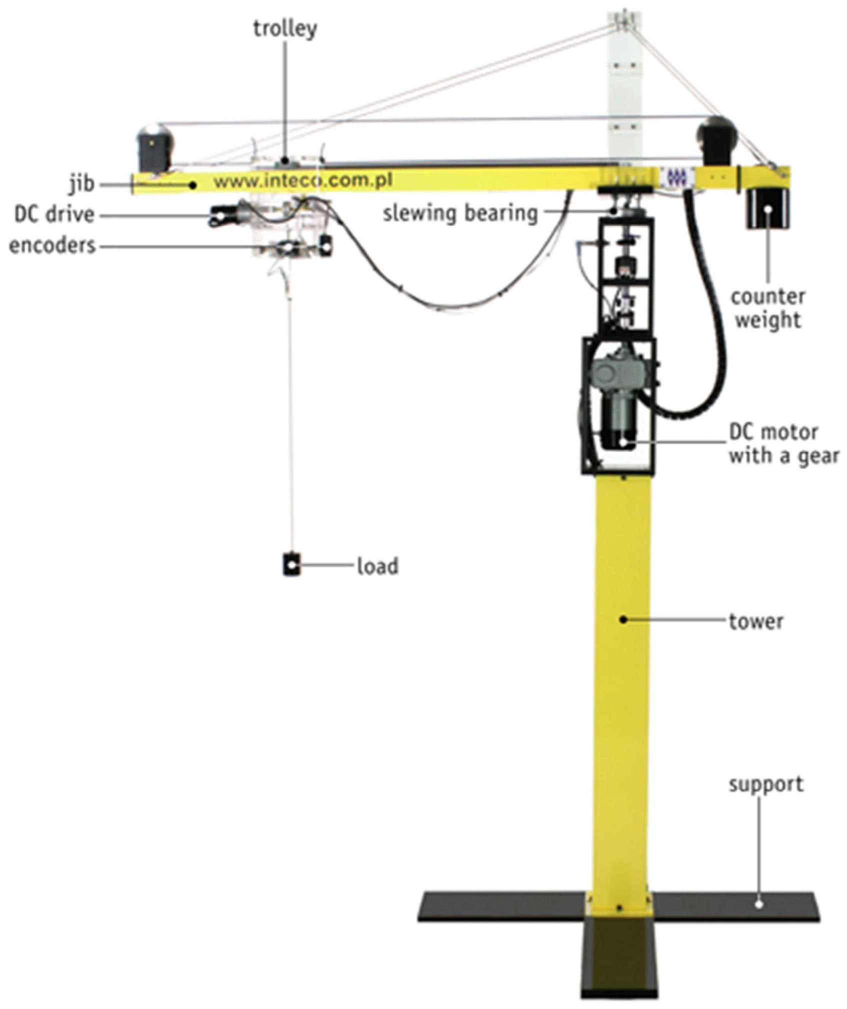


Figure 1. *Tower Crane* [1]

The system has 5 degrees of movement (DOF), with 2 axis and 3 angles. Vertical *axis z* along which, the cargo is pulled up or down, and the arm/jib is rotated by the *Theta angle*. Horizontal *axis y* along which, the trolley is moved. And *angles X*, *Y* corresponding to the *angle* between the projection of the lift-line onto the *xz* plane and the lift-line.

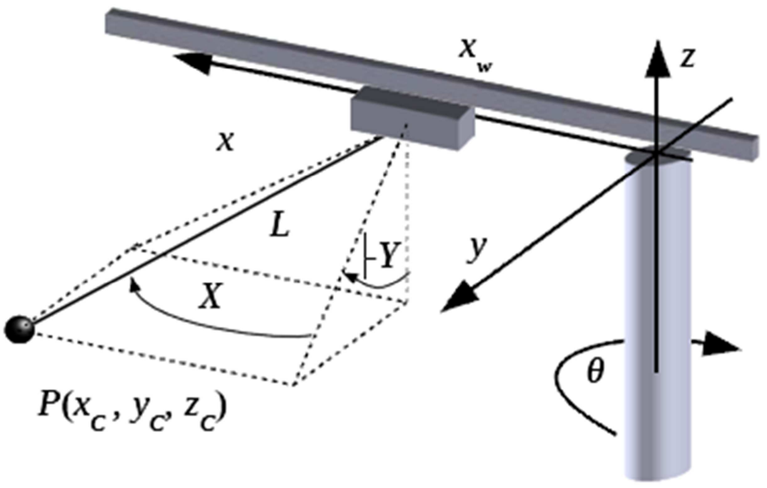


Figure 2. *Degrees of movement* [2]

The PC equipped with the RT-DAC/USB multipurpose digital I/O board communicates with the power interface. The whole logic necessary to activate and read the encoder signals and to generate the appropriate sequence of pulses of PWM to control the DC motors is configured in the Xilinx chip of the RT-DAC/USB board. All functions of the board are accessed from the Tower Crane toolbox which operates directly in the MATLAB/Simulink environment [2].

2.2 Microsoft Kinect for Windows

Kinect is a line of motion sensing input devices by Microsoft for Xbox 360 and Xbox One video game consoles and Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands [3].

Figure 3. *First generation Kinect for Windows* [16]

Kinect uses 2 cameras – one for a color stream, the other for infrared stream. There is also an infrared emitter, which projects an infrared grid into the area of 1.2 – 3.5 meters. Finally, there is an array of 4 microphones for noise negation and voice commands.

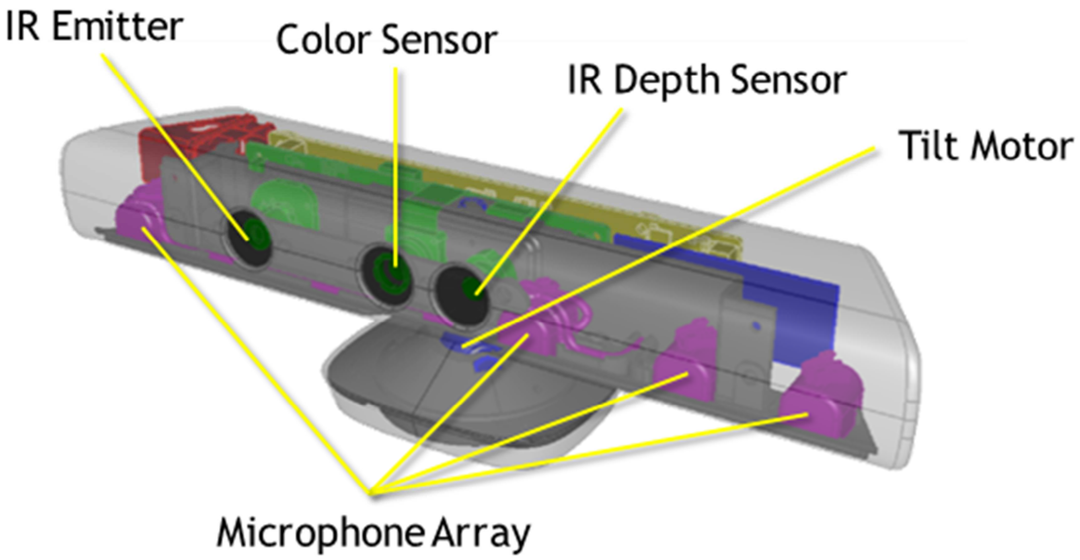


Figure 4. *Kinect main hardware* [16]

2.3 MATLAB

MATLAB is a numerical computing environment and fourth-generation programming language. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and FORTRAN [4].

2.4 Simulink

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems [5].

Simulink is the main environment in this work, which acts as a layer between the control application and the physical model. Its role is to receive formatted data from Kinect and convert it to signals that the crane can understand.

2.5 Real-Time Windows Target

Real-Time Windows Target provides a real-time engine for executing Simulink models on a Windows PC and blocks that connect to a range of I/O boards. It helps to create and control a real-time system for rapid prototyping or hardware-in-the-loop simulation [6].

2.6 Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment from Microsoft. It is used to develop console and graphical user interface (GUI) applications along with Windows Forms or WPF applications. Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer [7].

The control application was built from scratch and fully debugged in Visual Studio, using C++ for internal logic realization and .NET for graphical user interface.

3. Development stages

3.1 Microsoft Developer Network and Kinect code examples

The Microsoft Developer Network (MSDN) is the portion of Microsoft responsible for managing the company's relationship with developers and testers, such as: hardware developers interested in the operating system, and software developers utilizing various OS platforms or using the API and/or scripting languages of Microsoft's applications [8].

The main branches of MSDN are very informative with a lot of information about specific topic, useful examples and constant user help and input. One could say, that a beginner in programming can find there almost every type of information he will need to develop software, or to find help on obscure topic.

The above statement does not apply to the Kinect branch of the site. Although there is a lot of documentation for Kinect functions for C++, C# and Visual Basis languages, it lacks with examples. Especially for C++, since Microsoft is trying to push C# as a better alternative.

Because of learning a new language is usually time consuming and the problems listed above, it was decided to find another way to understand C++ programming for Kinect. That is why a big beginner example of C++ code for an old version of Kinect Source Development Kit was found. The only problem was that all of the commentary and explanations were in Japanese [9].

By reading and testing the example code line by line, while also referring to MSDN specifications a simple console based application was written. It could track user’s right shoulder and wrist coordinates, and by using trigonometry, output users arm angle relative to the floor.

3.2 Serial port

The next stage was to learn about serial communications, and their emulation. It was decided that this type of connection between Simulink and the application, because of simplicity, adequate speed and possibility to use ether virtual or physical COM port.

Again, referring to the main branch of MSDN and internet, a simple application was built, that wound send a formatted string to the COM port. At this point, the final format of the message was already decided.

After that the code for arm tracking and string sending was combined and tested, with a terminal as the receiver of formatted data. Problems occurred, when real crane testing began. There was a big problem with buffering and delays, the crane could not be controlled in real time. The idea was scrapped and a new solution was proposed.

It should be said, that a simple workaround for delays and buffering was found later in development – increase the sampling frequency of Simulink's COM port.

3.3 Simulink Support for Kinect

At Matlab central file exchange a ready solution was found – Simulink Support for Kinect. It had all of the functionality needed, fully built in Simulink environment, with extensive documentation. A virtually working model was assembled.

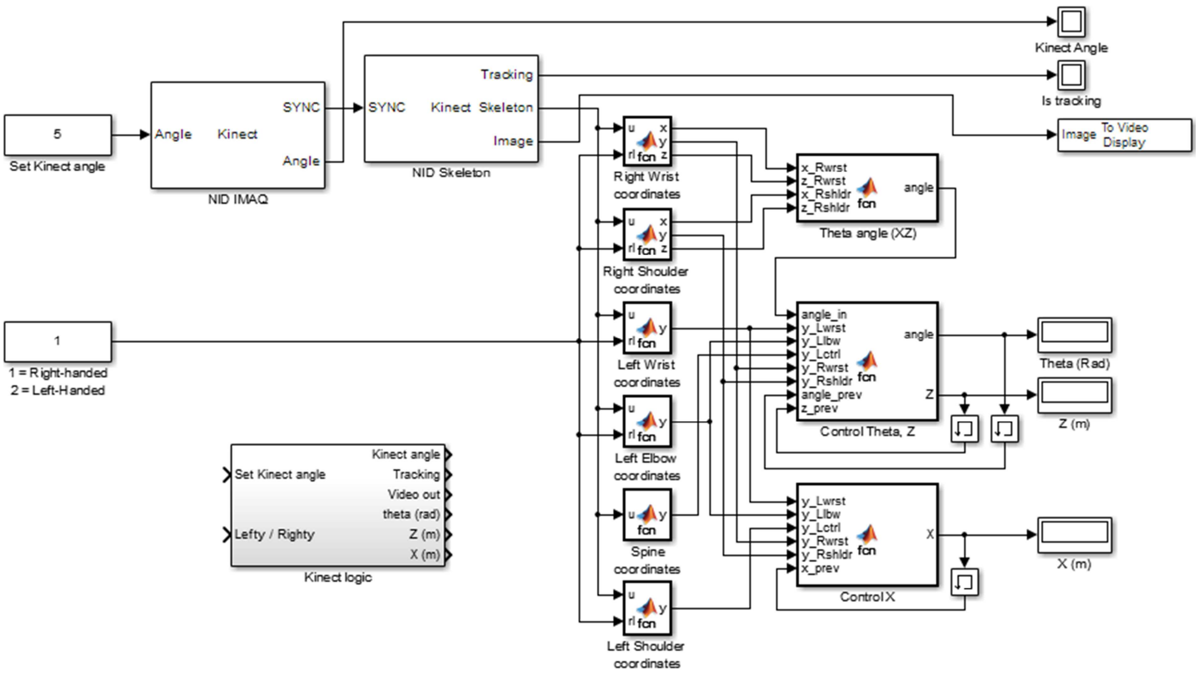


Figure 5. *Simulink for Kinect model*

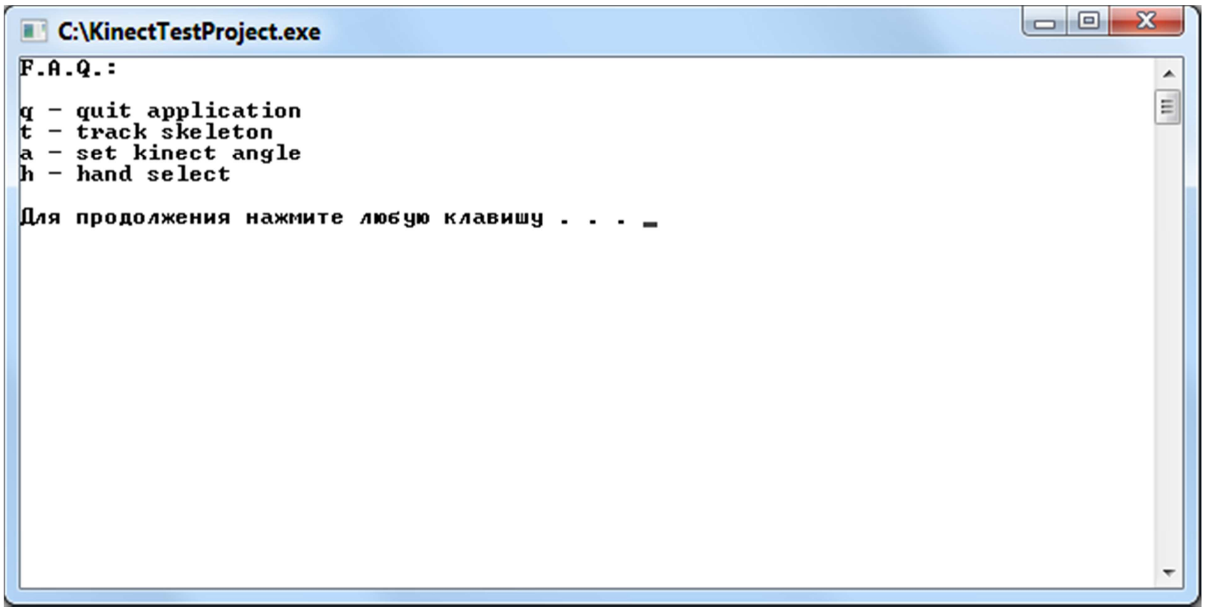
The seemingly perfect design had a huge flaw – Real-time Windows target cannot work along with any application using standard C++ header called Windows.h.

Windows.h is a Windows-specific header file for the C/C++ programming language which contains declarations for all of the functions in the Windows API, all the common macros used by Windows programmers, and all the data types used by the various functions and subsystems. It defines a very large number of Windows specific functions that can be used in C/C++ [10].

Windows.h is also a part of any Kinect application written with C++ and since the Simulink support was written in the same language the second iteration of the work was also scrapped. Since an a attempt to re-write the code in a way, that all needed functions and classes from Windows.h would be isolated, most likely would take too much time.

3.4 Console version Kinect t-crane application

Model shown in figure 5 was programmed as a console application. It could do everything that the GUI version could, just without a user friendly interface and exception handling.



```
F.A.Q.:

q - quit application
t - track skeleton
a - set kinect angle
h - hand select

Для продолжения нажмите любую клавишу . . .
```

Figure 6. *Console Kinect t-crane application*

4. Modeling and setup

4.1 Simulink Model

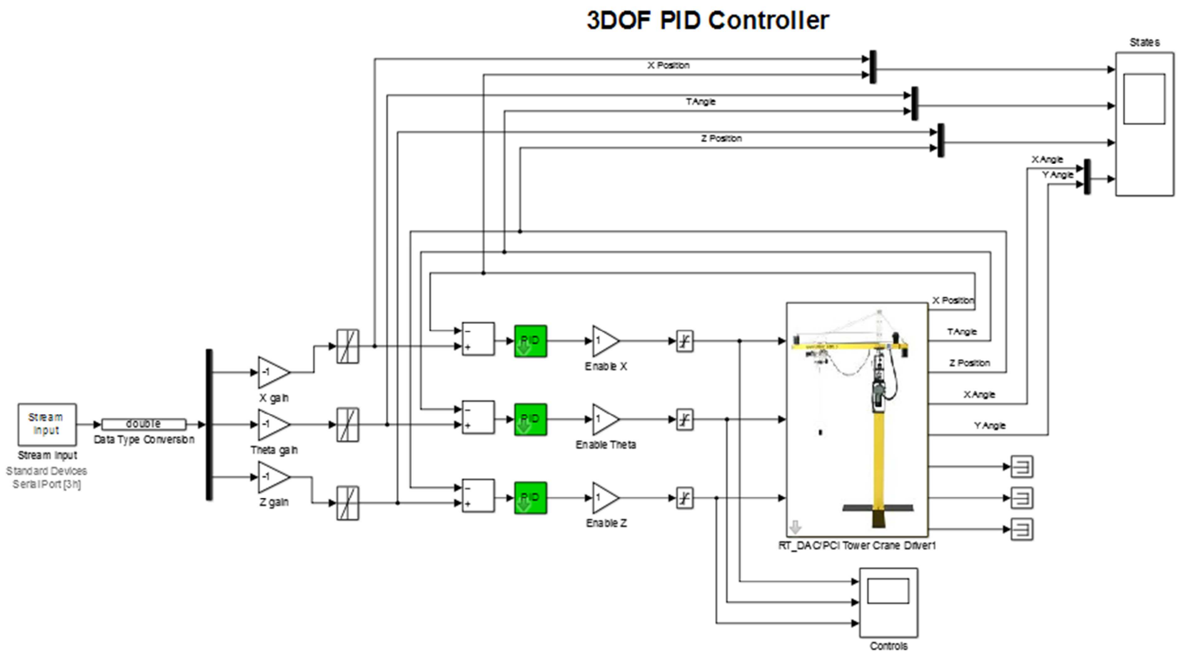


Figure 7. Simulink model

The template for this model was taken from a 5DOF demo provided by manufacturer INTECO. Stream input block was added and configured with the appropriate settings. And the system was downgraded to 3 degrees of movement (arm rotation, trolley position, cargo height), because of the problems related to the feedback loop, which were not a part of the original project goal.

The downgrade was needed because of cargos high momentum. As of current, the feedback was too unstable resulting in very rapid oscillation of the cargo. In the attempt to stabilize, the crane arm wound move rapidly, thus increasing the oscillation. A time consuming solution was implemented – angular speed of the crane arm was reduced. In result, if operators hand movement is very fast, the crane will respond much slower, but with the intended precision and higher stability.

4.2 Stream input settings

A complete user manual for default stream input settings could be found in the Appendix of this thesis.

4.3 Finalizing the setup

After applying any changes to the Simulink model, it must re-compiled/re-built, by using the toolbar menu *Code* -> *C/C++ Code* -> *Build Model* or by *Ctrl + B* keyboard shortcut.

If the build was successful, the model needs to be connected to the target via toolbar menu *Simulation* -> *Connect To Target.* After that, the model should be run by using a keyboard shortcut Ctrl + T or by toolbar menu *Simulation* -> *Run*.

After completing the steps above and physically starting the crane, the setup is complete and ready to use.

5. Application

5.1 Requirements

Internal logic requirements:

- From the developer stand point; easy and readable code with adequate commentary, no "spaghetti" code and consistent code without hacks and undocumented capabilities
- Safety critical execution. If the crane operator leaves Kinect field of view or does unexpected actions the system must pause or stop completely
- Exception handling. If Kinect is not found, COM port is not accessible or any other problems occur, the application should warn about these failures

Graphical user interface requirements:

- Easy to understand, simple to use
- On-screen help
- Nothing unnecessary or redundant, thus compact and could be run on low resolution monitors
- Start/Stop buttons, along with Connect/Disconnect buttons.
- Reset position button
- Set Kinect angle button
- Selection of dominant hand

System requirements:

- Virtual of physical COM ports
- Crane full control by user gestures
- Relatively fast Windows 7 and above based PC for simultaneously handling Kinect and Real-time Windows Target calculations. Ideally a 2-core or more processor, running at 2.8 GHz or higher, with 2 GB of RAM or higher

5.2 Interface realization

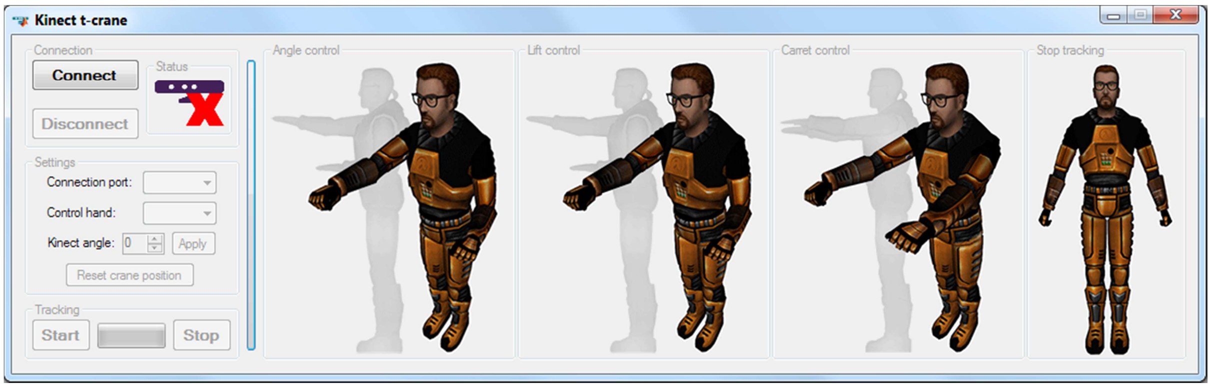


Figure 8. *Expanded interface*

The GUI colors correspond to selected Windows theme. All of the controls are from standard Windows templates. Control help images are animated and hidden by default.

The main problem of converting a console controlled application to graphical user interface control was the threading problem. While the skeleton tracking realization in console was started by pressing a key on a keyboard, and stopped by assuming a certain pose/gesture, GUI realization would not work this way because of the stop button. If tracking were started, it could not be stopped with a press of a button, because application would permanently freeze in infinite cycle of scanning and sending data. Assuming the stop pose/gesture would still work, but an alternative of a stop button was still needed. Because of that multi-threading needed to be used.

By referring to Microsoft Developer Network a few threading solutions were found [11][12]. The problem of managed threading was that it was too complicated to understand and would not work perfectly in its simplest realization with .NET. It generated too many crashes and memory leaks. The alternative was a standard background worker class from Visual C++/.NET libraries. The result is a stable and simple implementation of infinite loop, that can be ended by a press of a button.

Full user manual can be found in the appendix of this thesis.

5.3 Logic realization

Internal logic was realized as big Class consisting of several functions.

- `int resetCrane(int iIndex)` – this function sends a string with default crane coordinates, using the *`WriteComPort`* function, using the selected port as an argument
- `void killThread()` – this function terminates the scanning and sending data loop, that runs in a dedicated thread
- `int InitKinect()` – this function start the Kinect with needed flags and smoothing parameters
- `void ShutdownKinect()` – this function stops the Kinect
- `void SetKinectAngle(int KinectAngle)` – this function sets Kinect angle, using user inputted angle as an argument
- `char *SelectCOM(int iIndex)` – this function is used by *`WriteComPort`* function to interpret user selected COM port, using the selected port as an argument
- `int WriteComPort(char* cData, char* cCOMPort)` – this function sends data to selected COM port, using formatted data and selected port interpretation from *`SelectCOM`* as arguments
- `int CheckComPort(char* cCOMPort)` – this function check availability of selected COM port, using selected port interpretation from *`SelectCOM`* as arguments
- `void ResetCoodinates()` – this function sets the current and buffered coordinates to zero
- `int Logics(int bHand, int iIndex)` – this is the main function, that handles raw data from Kinect, manipulates and formats it to a usable string. Arguments are dominant hand and selected COM port

6. Conclusion

When beginning this work a few months ago, the initial idea and goal were very simple, without any visible pitfalls. But the development of such system and application proved otherwise, with many problems emerging with every new developments stage.

Starting with very technical Kinect documentation from Microsoft, and scarce amount of programming examples. Moving to highly detailed code, but with Japanese commentary. And after that, moving back and forth from implementing the controls from scratch or using pre-made working solution, were not trivial tasks. Though, very useful in understanding the specifics of programming for Kinect, and for developing these kind of projects. Showing what kind of work may be present in real life companies that develop such control systems.

The goals of the work were met. The result is a human machine interface, with relatively short code, around 1000 lines including whitespaces. Running on Windows PC, and connecting to MATLAB via COM port. With a simple design and a potential to be used with any other device, similar to a tower crane, and not dependent on MATLAB software.

Within this work aspects of system feedback were not sufficiently detailed. Further research in finding optimal settings of proportional-integral-derivative controllers or switching to another type of control loop would further enhance the usefulness and application of this kind of motion control methods.

Further application development would most likely consist of switching from Kinect Source Development Kid to open sourced development platform, like OpenKinect project. Ultimately, porting the HMI to other platforms – Linux and Mac OS X.

This application is a good example how Kinect can be used not only in games, but also in industrial and laboratory conditions and will benefit anyone who is looking for ways to use MATLAB/Simulink with non-standard external hardware.

List of used materials

[1] Inteco LLC, “Tower Crane”. [Online]. Available: http://www.inteco.com.pl/index.php?option=com_content&view=article&id=10&Itemid=17 [Accessed 17.01.14].

[2] Inteco LLC, “Tower Crane User’s Manual”. [Online]. Available: http://www.inteco.com.pl/attachments/article/10/TowerCrane_s.pdf [Accessed 17.01.14].

[3] Wikimedia Foundation Inc, “Kinect”. [Online]. Available: http://en.wikipedia.org/wiki/Kinect [Accessed 17.01.14].

[4] Wikimedia Foundation Inc, “MATLAB”. [Online]. Available: http://en.wikipedia.org/wiki/Matlab [Accessed 17.01.14].

[5] The MathWorks Inc, “Simulink”. [Online]. Available: http://www.mathworks.se/products/simulink/ [Accessed 17.01.14].

[6] The MathWorks Inc, “Real-Time Windows Target”. [Online]. Available: http://www.mathworks.se/products/rtwt/ [Accessed 17.01.14].

[7] Wikimedia Foundation Inc, “Microsoft Visual Studio”. [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio [Accessed 17.01.14].

[8] Wikimedia Foundation Inc, “Microsoft Developer Network”. [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Developer_Network [Accessed 17.01.14].

[9] Natural Software, “Kinect for Windows SDK v1.7 の Interactionを使ってみる(C++)”. [Online]. Available: http://www.naturalsoftware.jp/blog/8220 [Accessed 17.01.14].

[10] Wikimedia Foundation Inc, “windows.h”. [Online]. Available: http://en.wikipedia.org/wiki/Windows.h [Accessed 17.01.14].

[11] Microsoft Developer Network, “Managed Threading”. [Online]. Available: http://msdn.microsoft.com/en-us/library/3e8s7xdd(v=vs.110).aspx [Accessed 17.01.14].

[12] Microsoft Developer Network, “BackgroundWorker Class”. [Online]. Available: http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker(v=vs.110).aspx [Accessed 17.01.14].

[13] Microsoft Developer Network, “Serial Communications in Win32”. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms810467 [Accessed 17.01.14].

[14] Microsoft Developer Network, "KinectInteraction". [Online]. Available: http://msdn.microsoft.com/en-us/library/dn188679.aspx [Accessed 17.01.14].

[15] Wikimedia Foundation Inc, "COM-порт в Windows (программирование)". [Online]. Available: http://ru.wikibooks.org/wiki/COM-порт_в_Windows_(программирование) [Accessed 17.01.14].

[16] Microsoft Developer Network, "Kinect for Windows Sensor". [Online]. Available: http://msdn.microsoft.com/en-us/library/hh855355.aspx [Accessed 17.01.14].

Acknowledgments

I would like to express my very great appreciation to Valve Corporation for using their Gordon Freeman model from Half-Life series for help illustrations. And wish not to infringe any copyright as stated in fair use model for creative work.

I would also like to thank the developer of kHED modeling software (http://khed.glsl.ru/) for using his editor to animate help illustrations of the application.

My grateful thanks are also extended to users of HLFX community and forum (http://hlfx.ru/forum/) for helping me to understand threading technology in .NET and C++.

Finally, I wish to thank my close friend Igor Petrov for helping me to understand and showing the specifics of programming for C/C++ languages.

Appendix 1. User manual